Assignment 1 PWM Operation and Latency Measurement in Zephyr RTOS (150 points)

In this assignment, you are requested to implement a program that adds 3 shell commands in Zephyr RTOS (version 1.14.2) running on Galileo Gen 2 board:

1. project1 RGB-display x y z
2. project1 int-latency n
3. project1 cs-latency n

where "project1" is the root command, "RGB-display", "int-latency" and "cs-latency" are subcommands, and "x", "y", "z" and "n" are command arguments.

All your project files should reside in the directory "zephyr/samples/project1_nn" of Zephyr source tree where nn is the last 2 digits of your ASU id. There should be no changes to the original Zephyr source (except the patches applied to gpio_dw.c, pwm_pca9685.c, and pinmux.c).

1. project1 RGB-display x y z

   In the example program "test-led", red, green, and blue leds are turned on and off to display 8 colors (from none to white) sequentially every second. To add intensity control when a led is on, we can apply a PWM signal of variant duty cycles to the led. On Galileo board, a PCA9685 device can be programmed to put out PWM signals and its driver can be enabled in Zephyr.

   Your main program should have an endless loop to control the RGB led display. You can modify "test-led" program such that when a led is on, a PWM with specific duty cycle is sent to the led. We will assume PCA9685's PWM3, PWM5, and PWM7 are connected to the R, G, and B pins of the led. The shell command sets the duty cycles of the PWM signals to *x*, *y*, and *z*. For instance, "project1 RGB-display 20 40 60" modifies the duty cycles of RGB PWM signals to 20%, 40%, and 60%, respectively.

2. project1 int-latency n
3. project1 cs-latency n

   Two important performance metrics of RTOS are interrupt latency and context switching overhead. Interrupt latency is the total delay between the interrupt signal being asserted and the start of the interrupt service routine execution. This delay may be extended if an interrupt arrives when the RTOS is in a non-preemptive critical region. As for context switching overhead, it is the delay of context switching process of saving the context of the executing thread, restoring the context of a ready thread, and starting the execution of the ready thread.

   The two shell commands start to measure interrupt latency and context switching overhead of Zephyr RTOS (version 1.14.2) on Galileo Gen 2 board. To generate interrupts, you can loop back a gpio output signals to a gpio input pin which has interrupt enabled at rising or falling edge. To trigger a context switch, you can let a thread unlock a mutex for which a thread of higher priority is waiting. To record the instants of event occurrences, you can read x86's Time Stamp Counter (TSC) which provides a much better granularity than typical OS timer. The command argument "*n*" indicates the number of measurement samples to be collected and, when this is done, the average latency is reported (printed on the console). In addition, we will assume that:

   - gpio pins 6 and 7 of GPIO_0 device are used for loopback output and input gpio signals, respectively.
   - No new measurement command will be accepted if a previous measurement is still ongoing.

Note that some operations performed in the measurement may introduce delays and cause the measured elapsed time inaccurate. For instance, you may “read timestamp”, “write 1 to gpio6”, and “read timestamp” in the interrupt callback function for gpio7. The elapsed time between the two timestamps includes the delay of “write 1 to gpio6”. A careful design of measurement approach should be considered to remove these delays.

Due Date

The due date is 11:59pm, Feb. 15.

What to Turn in for Grading

- Compress all files and subdirectory of “zephyr/samples/project1_nn” into a zip archive file named RTES-LastName-FirstInitial_01.zip and submit the zip archive to Canvas by the due date. Note that any object code or temporary build files should not be included in the submission.
- Please make sure that you comment the source files properly and the readme file includes a description about how to make and use your software. Don’t forget to add your name and ASU id in the readme file.
- There will be 20 points penalty per day if the submission is late. Note that submissions are time stamped by Canvas. If you have multiple submissions, only the newest one will be graded. If needed, you can send an email to the instructor to drop a submission.
- The assignment must be done individually. No collaboration is allowed, except the open discussion in the forum on Canvas. The instructor reserves the right to ask any student to explain the work and adjust the grade accordingly.
- Failure to follow these instructions may cause deduction of points.
- Here are few general rule for deductions:
  - No make file or compilation error -- 0 point for the assignment.
  - Must have “–Wall” flag for compilation -- 5-point deduction for each warning.
  - 10-point deduction if no compilation or execution instruction in README file.
  - Source programs are not commented properly -- 10-20-point deduction.
- ASU Academic Integrity Policy (http://provost.asu.edu/academicintegrity), and FSE Honor Code (http://engineering.asu.edu/integrity) are strictly enforced and followed.

How your program will be graded:

1. Copy your zip file to zephyr/samples/project1_nn
2. Unzip your submission
3. Mkdir build, cd build, cmake, and make
4. Run your application image on Galileo Gen 2 board
5. Examine your source code.